

CIS5930: Fundamentals of High Performance Optimization

- Low-level code optimization
 - Use actual machines & kernels
 - Roll your own timers
 - Learn assembly
- Application of architecture
- Very hands-on
- **Instructor:** Dr. Whaley
 - **Office** : LOV 167
 - **email** : whaley@cs.fsu.edu
- **Room:** LOV 103
- **When:** Mon & Wed 11AM - 12:15PM.
- **text:** Web/man/hand
- **Prereq:** Comp Arch, Low-lev prog

CIS5930 – Grading

With this few students, will probably grade the course a little more loosely:

- 85% projects
 - 0-2 presentations/student
 - Mostly programming projects
- 10% oral/written quiz
- 5% class participation
- No written final

Don't go there:

- 60% prog projects, including final project
- 35% written exams, including final exam
- 5% class participation

Initial Topic Overview

1. Real world tester/timer creation
2. Architecture overview
 - Memory: disk, TLB, caches, registers
 - FPU: pipelining, ILP
3. Optimization overview
 - Memory: data copy, blocking, prefetch, fetch scheduling, efficient array indexing
 - FPU: software pipelining, superscalar sched, scalar exp
4. Basics of assembly programming
 - AT&T x86 (maybe x86-64)
 - Perhaps SPARC/PowerPC
5. SSE, prefetch
6. Possibly:
 - (a) FP representation & IEEE
 - (b) HPC arch overview
 - (c) pthreads
 - (d) MPI

Basic Definitions

- **MFLOPS**: millions of floating point operations per second
 - FLOPS measure of speed, requires less info than time
- What is FPU peak of machine?
 - $n_{fpu} * (\text{flops/cycle}) * \text{Mhz}$
 - usually $1-4 * \text{Mhz}$
 - Do timing results make sense?
 - * $0 < mf < peak$ & repeatable?
- **vector**: 1-dimensional array
- **matrix**: 2-D array (details later)
 - One dimension contiguous (vector)
 - Other dimension constantly strided
 - I usually do column-major arrays (good ol' F77)
 - `**p != p[100][100] != above`

Timing Basics

- Two types of timing clocks:
 1. CPU time: time CPU uses for process
 - Low resolution, little interference
 - Run mult times, take median
 2. Wall time: elapsed time in real world
 - High resolution, subject to interference
 - Required for parallel timings
 - Run mult times, take minimum
- What if timings not repeatable?
 1. Use higher res timer
 2. apply timer-specific resolution trick
 3. Time multiple invocations of kernel
 - Repeat call until 0.5 sec of peak MFLOP rate is achieved
- Considerations for multiple invocations:
 - Over/under flow, or bad data?
 - Is result used enough to fool compiler?
 - Are caching affects handled?

Example Vector Summation Timer

```
t0 = my_time();
sum = vecsum(N, X);
tim = my_time() - t0;
```



```
nrep = (mflop*1000000.0 + N-1)
      / (1.0*N);
if (nrep < 1) nrep = 1;
t0 = my_time();
for (i=0; i < nrep; i++)
    sum = vecsum(N, X);
tim = my_time() - t0;
```



```
t0 = my_time();
for (i=0; i < nrep; i++)
    sum += vecsum(N, X);
tim = my_time() - t0;
```



```
t0 = my_time();
for (i=0; i < nrep; i++)
    if (((i>>1)<<1) == i)
        sum += vecsum(N, X);
    else
        sum -= vecsum(N, X);
tim = my_time() - t0;
```

Basic Analysis Definitions

- Number of FLOPS in algorithm
- Number of references in algorithm
- Number of data words
 - Possible reuse when $\#$ of ref $>$ $\#$ of data words

Example Operations

DDOT: dot product of vectors X & Y

```
for (dot=0.0, i=0; i < N; i++)  
    dot += x[i] * y[i];
```

- $2N$ FLOPS
- $4N$ mem ref
- $2N$ data

DGEMM: matmul $C = A \times B + C$

```
for (j=0; j < N; j++)  
    for (i=0; i < N; i++)  
        for (k=0; k < N; k++)  
            C[i+j*ldc] +=  
                A[i+k*lda] * B[k+j*ldb];
```

- $2N^3$ FLOPS
- $4N^3$ mem refs
- $3N^2$ data

Homework time!

